De-anonymizing the Internet Using Unreliable IDs

Yinglian Xie, Fang Yu, and Martín Abadi
Microsoft Research Silicon Valley

ABSTRACT

Today's Internet is open and anonymous. While it permits free traffic from any host, attackers that generate malicious traffic cannot typically be held accountable. In this paper, we present a system called HostTracker that tracks dynamic bindings between hosts and IP addresses by leveraging application-level data with unreliable IDs. Using a month-long user login trace from a large email provider, we show that HostTracker can attribute most of the activities reliably to the responsible hosts, despite the existence of dynamic IP addresses, proxies, and NATs. With this information, we are able to analyze the host population, to conduct forensic analysis, and also to blacklist malicious hosts dynamically.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General—*security and protection*; C.2.3 [**Computer Communication Networks**]: Network Operations—*network management*

General Terms

Measurement, Security

1. INTRODUCTION

The Internet is designed to be open and anonymous. Just by obtaining an IP address, a host can easily connect to the Internet and freely talk to other hosts without exposing its real identity. This open and anonymous architecture, which enabled the Internet to expand quickly, is also the source of security concerns. Attackers can easily hide their real identities behind IP addresses. Dynamic IP address assignment [10] poses challenges to the commonly used IP-based approach to detect, blacklist, and block malicious traffic. When an attacker changes its IP address, legitimate activities that subsequently use the old IP address will be misclassified as bad, while malicious activities from the new IP address will slip through. The numerous proxies and NAT devices also imply that blacklisting can result in denial of service to many legitimate clients that share IP addresses with attackers.

SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.

Security rests on host accountability. By accountability we mean the ability to identify the hosts responsible for traffic [18], which is typically the basis for punishing misbehavior. Several clean-slate solutions have been proposed to provide accountability in the Internet, thus eliminating the problems created by dynamic IP addresses, proxies, and NATs. For example, Accountable Internet Protocol (AIP) [1] changes the IP layer by utilizing self-certifying addresses to ensure that hosts can prove their identities. Although these proposals are attractive, they are hard to deploy and many service providers need immediate means to combat network attacks.

In this paper, we aim to develop an immediate, practical approach to associate traffic with hosts. We revisit the utility of IP for accountability and seek to understand its limitations. In particular, we aim to quantify the ability to infer the bindings between a host and an IP address (which we call host-IP bindings). Our study attempts to answer the following two questions:

- To what extent can we use IP addresses to track hosts?
- Can we use the binding information between hosts and IP addresses to strengthen network security?

The answers to these questions have implications in numerous security applications. First, the ability to identify a host over time permits attack investigation to trace back to the start of malicious activities and to reveal previously undiscovered ones. Second, it facilitates the task of building more accurate blacklists to both monitor and block ongoing attacks, and prevent new attacks in the future. Furthermore, the knowledge of compromised hosts detected by one application can potentially be shared to benefit other applications, as botnets may be rented out to different attackers over different periods.

We present *HostTracker*, a system that relies on application-level events to automatically infer host-IP bindings. In today's Internet, establishing accurate host-IP bindings is challenging: malicious activities, dynamic IP addresses, proxies, and NATs make it difficult to differentiate activities from distinct hosts. HostTracker leverages IDs derived from application-layer logs in order to create unique host identifiers and to track the bindings of hosts to IP addresses over time. For our study, we employ a month-long user login trace collected at a major Web email service provider. Our findings include:

- We show that, even without built-in host identities, using IP addresses, anonymized user IDs, and their associated events, we can track a large percentage of host activities with high accuracy. Overall, 76% of the events in the application log can be attributed to hosts, and 92% of hosts can be tracked correctly. This result is consistent across many IP-address

ranges, suggesting that tracking host-IP bindings is widely applicable.

- As an application, relying on a set of previously detected botnet email accounts (5.6 million) [38], we can apply the host-IP binding information to identify the compromised hosts and many additional botnet email accounts (12.6 million) with a low false positive rate (0.4%).
- We can also build a host-aware blacklist using the host-IP binding information. This blacklist can help us follow the trail of malicious hosts in order to block attacks (from 20.8 million malicious accounts) in real time without significantly affecting normal user access (with a 0.1% false positive rate).

With this initial evidence, we believe that tracking host-IP bindings holds promise to improve network security in the current Internet. It opens up new opportunities to re-think many existing attack detection and prevention mechanisms, particularly those that use blacklists to block traffic.

The goal of our work is not to develop bullet-proof accountability schemes, but instead to support accountability well enough to serve the needs of applications. In this sense, our approach is complementary to clean-slate proposals. It might be part of a temporary solution until architectural changes can be made to the networking stack, and it might inform the eventual design of those changes.

Despite the value of accountability, we believe that anonymity also has a legitimate place in the Internet (for instance, for enabling forbidden communication in repressive countries). Our findings suggest that "low-tech" anonymity is more fragile than one might have expected. Nevertheless, anonymity can often be achieved with sophisticated tools (e.g., onion routing [31], anonymous remailers, etc.). Those tools do not negate the value of HostTracker for many applications. Further, we hope that our work will contribute to the ongoing debate on the balance between accountability and anonymity in the next generation of IP networks.

2. RELATED WORK

Host accountability in the Internet has long been a topic of substantial interest. A large body of previous work has focused on providing source accountability to identify the true network origin of traffic. Both stepping-stone techniques and source-address spoofing are commonly leveraged to hide attacker identities. A few early efforts have proposed solutions to detect stepping-stone attacks by packet-timing analysis and content analysis [37, 9, 5, 33]. Ingress and egress filtering [12, 16], which have been partially deployed, can prevent source-address spoofing. Other proposed approaches also require changes to the existing routers or the routing infrastructure. Among them, IP-traceback techniques [29, 19, 28, 3, 6] aim to determine the source(s) of packets by storing additional state at routers or marking packets along their paths. Passport [20] validates host source addresses using cryptographic keys.

Many of today's attacks require establishing TCP connections, in which source-address spoofing is difficult. On the other hand, botnet hosts do not need to spoof IP addresses: the transient nature of the attacks and the dynamics of IP-address assignment make it difficult to pinpoint the exact compromised host entities as their IP addresses change. To offer host accountability as a fundamental security property, Accountable Internet protocol (AIP) [1] is a clean-slate design that uses self-certifying addresses to ensure that hosts and domains can prove their identities without relying upon a global trusted authority.

While clean-slate proposals offer attractive long-term solutions, our work targets today's Internet to mitigate security threats until architectural changes can be deployed globally. With the rapid advance of attack detection techniques (e.g., [25, 26, 14, 13, 36]), we focus on the ability of tracking host-IP bindings so that both proactive and reactive measures can be taken once a compromised host is identified by its IP address at a certain time.

In order to block unwanted traffic from compromised hosts, various blacklists are widely used (e.g., [11, 30, 4]). However, since these blacklists all represent hosts by IP address, it remains unclear how to apply them effectively in the presence of dynamic IP addresses and proxies. (Recent studies have shown that a significant fraction of the Internet IP address space is dynamic [35] and that the number of proxies and NATs is non-trivial [7].) By inferring the host-IP bindings, HostTracker potentially provides new possibilities of applying IP blacklists in a more effective way.

More broadly, botnet detection and defense has attracted a lot of attention. Botnet spamming attacks are commonplace. In many of these, each bot host is used to set up a spam email server [8, 24, 2, 15, 35]. Recently, a new botnet spamming attack signed up tens of millions of free Web email accounts for sending spam emails [32]. Applying a blacklist-based approach to this new attack is even more challenging, as legitimate users and attack accounts may log in from a common host.

Finally, remote device fingerprinting is also an active area of research, where several proposals leverage packet-level traffic characteristics or clock skews of a host to generate OS or device fingerprints (e.g., [22, 21, 17]). Exploiting subtle hardware differences can potentially be quite robust, but the accuracy of the existing techniques currently prevents them from being deployed at a large scale. Our approach offers an alternative solution to hardware fingerprinting—application-level IDs might be regarded as host fingerprints (with noise).

3. PROBLEM FORMULATION

Host accountability is often a key component associated with auditing and forensic analysis. We distinguish host accountability from host authentication. Whereas host accountability may only be "after the fact", host authentication typically establishes a trusted identity before any further traffic or requests from the corresponding entity can be received or processed. Accordingly, in this section, we consider the problem of tracking hosts relying on traces of past application-level events.

3.1 Host-Tracking Graph

Given that an IP address is the only directly observable identifier of host network traffic in the current Internet and that a host's IP address can dynamically change, our goal is to perform a posteriori analysis of traffic (e.g., network activity or application request) logs and generate a *host-tracking graph* to infer the bindings between hosts and IP addresses over time.

Figure 1 depicts an example host-tracking graph for three different hosts `A`, `B`, and `C`. Time advances from left to right, with each rectangle representing the binding between a host and an IP address across a time range. In this example, host `A` was bound to IP address IP_1 during time range $[t_1, t_2]$. Later, host `A` moved to IP_2 and IP_5 during time ranges $[t_3, t_4]$ and $[t_5, t_6]$, respectively. We call a *binding window* a time range during which a host is bound to an IP address. In this example, the binding window for `A` at IP_1 is $w = [t_1, t_2]$.

Formally, we define the host-tracking graph $G : H \times T \rightarrow IP$, where H is the space of all hosts on the Internet, T is the space of time, and IP is the IP-address space. For example, if host $h \in H$ is bound to IP address $IP_1 \in IP$ at time $t \in T$, we represent this binding as $G(h, t) = IP_1$. Similarly, we write $G(h, w) = IP_1$

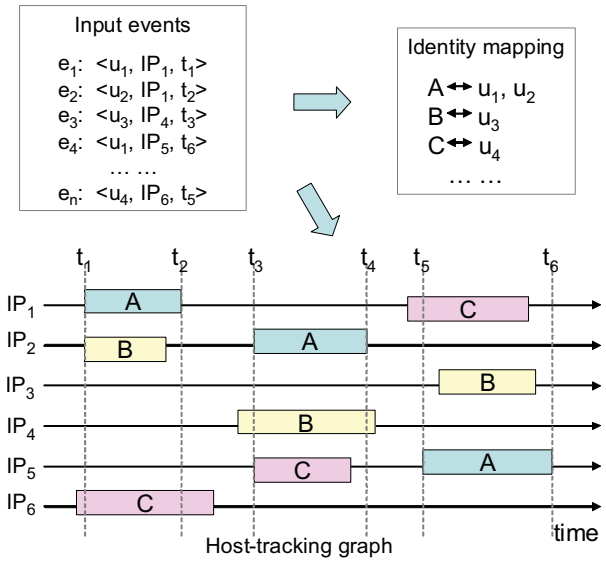

Figure 1: Generation of a host-tracking graph.

when $G(h,t) = IP_1$ for all $t \in w$. With this definition, we do not consider multi-homing hosts where a host may have different IP addresses concurrently. For proxies and NATs, different hosts may concurrently use one IP address. So it is possible to have $G(h_1,t) = IP_1$ and $G(h_2,t) = IP_1$, where $h_1 \neq h_1$. Note that the bindings from hosts to IP addresses may not be continuous, since hosts may not be up all the time and there may also exist unassigned IP addresses. For example, there are no hosts bound to address IP_3 before time t_5 in Figure 1.

With a host-tracking graph, we can attribute input events to the responsible hosts by finding the corresponding host-IP bindings. This enables us to reason about host accountability in two ways. First, for any network traffic originating from an IP address IP_i at time t_j, if there exists a single host h such that $G(h,t_j) = IP_i$, then h may be blamed for such traffic. Second, if host h is known to have bound to a different IP address IP_k at t_j, then we may conclude that h is not responsible for generating any traffic from IP_i at time t_j.

3.2 Host Representation

To derive a host-tracking graph, the first question is how to represent a host. Host representations are often application-dependent. In network security applications, it is desirable to represent a host as *an entire hardware and software stack* and track all its network activity.

Since we lack strong authentication mechanisms, we consider leveraging application-level identifiers such as user email IDs, messenger login IDs, social network IDs, or cookies. These identifiers are fundamentally unreliable in that they do not exactly correspond to hosts. On the other hand, each such identifier is typically associated with a human user. In many cases, such connections are helpful for tracking the corresponding principals. For example, a browser cookie appearing in requests from different IP addresses at different times *may* suggest that the corresponding host has connected to the Internet at different times with different DHCP IP addresses. Therefore, the group of user IDs that is associated with a host can essentially serve as a "virtual ID" for this host in order to keep track of the related activities. Figure 1 shows an example *identity-mapping table* that represents the mappings from unreliable IDs to hosts.

3.3 Goals and Challenges

Our goal is to generate the host-tracking graph using logs with unreliable IDs. As indicated in Figure 1, the input to our problem is a sequence of network or application events $e_1, e_2, \ldots, e_n$ over time. Each event contains three fields: *a unique unreliable ID*, *an IP address*, and the *timestamp* when we observe the ID being associated with the corresponding IP address. We would like to generate two outputs: the first being an identity-mapping table that represents the mappings from unreliable IDs to hosts, and the second being the host-tracking graph that tracks each host's activity across different IP addresses over time.

With the generated host-tracking graph, we refer to the set of input events that can be attributed to certain hosts as *tracked events*, and refer to the remaining events as *untracked events*. Correspondingly, we refer to those IDs that are used to represent hosts in the identify-mapping table as *tracked IDs*, and refer to the group of IDs that all map to one unique host as a *tracked ID group*.

Ideally, with a perfect host-tracking graph, every event is a tracked event. However, the huge amount of noise intrinsic in using unreliable IDs, the existence of dynamic IP addresses, NATs, and proxies, and the presence of malicious attack traffic all make the task of generating the host-tracking graph challenging:

- First, one user and hence one host may be associated with multiple IDs. Furthermore, a group of hosts can also share an ID if they are controlled by a common user.
- Second, a large fraction of the IP addresses in the Internet are dynamic and that their host-IP binding durations have huge variations [35].
- Third, proxies and NATs make it difficult to distinguish individual hosts behind these boxes [7]. Typically we cannot use a fixed set of IDs to represent a large proxy or NAT, as its user population may change frequently. Moreover, proxies and NATs may use dynamic IP addresses as well. Given the limitations of our dataset, we do not further distinguish hosts behind proxies and NATs. However, we do need to identify large proxies and NATs and take special actions in this case. Specifically, we treat proxies and NATs as hosts, and therefore their events are tracked.
- Finally, once a host is compromised, its traffic patterns may differ from those of normal hosts. For example, in recent Web-account attacks, attackers signed up tens of millions of free email IDs and then aggressively used them to send spam from compromised hosts. Those malicious IDs are not useful for identifying compromised hosts when we generate the host-tracking graph.

Therefore, realistic algorithms may generate only a subset of the real-world bindings. Moreover, not all host-IP bindings may be correctly identified. Our goal is to maximize the number of tracked events, while ensuring the host-IP binding inference accuracy.

4. TRACKING HOST ACTIVITIES

In this section, we present *HostTracker*—a system for tracking host-IP bindings by mining application-level activity logs. To simplify our description, in the rest of this section, we use user email IDs as our example application IDs and adopt user-login events as example input events.

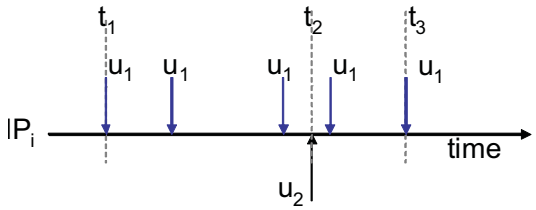

Figure 2: An example set of input events.

4.1 Overview

The host-tracking is performed separately for each IP range. The range information can be derived from the BGP table prefixes [27] or other sources such as the Whois database [34]. Usually, within a range, a user ID may bind to multiple IP addresses from a single host. For user IDs that appear across multiple ranges, the bindings to multiple IP addresses may also be caused by user mobility rather than host mobility. Therefore, HostTracker analyzes events from each range independently. In Section 5.4, we further study user mobility on a global scale.

In the host-tracking process, a crucial piece of information to recover is the identity-mapping table as shown in Figure 1, i.e., which user IDs map to which hosts. Without prior knowledge of which subset of user IDs can potentially be tracked, we regard the identity-mapping table as a hidden model that we would like to estimate. The high-level idea of HostTracker is to iteratively update our estimations so that it best fits the actual observation of the events from the input data.

Let us begin by assuming that each user ID would be a tracked ID group that maps to a unique host. Figure 2 shows an example input of six user-login events (represented as vertical arrows) from users u_1 and u_2 at IP$_i$. Here u_1 logged in five times from time t_1 to t_3. u_2 logged in at the same IP address at time t_2. In our dataset, we observed that the IP-address assignment in a large IP range was either random (e.g., [10, 23]) or static. In the random address assignment case, the probability of a host obtaining the exact same IP address in two different connections in a network range is usually very small. Therefore it is highly likely that the host corresponding to u_1 is bound to IP$_i$ throughout the duration between t_1 and t_3, including t_2. However, under the assumption that each user uniquely maps to a different host, there exists another host corresponding to u_2, which also binds to the same IP address IP$_i$ at time t_2. Our naive one-to-one identity mappings thus lead to an inconsistency, suggesting that we should re-estimate the mapping table.

To do so, we consider three possibilities: (1) multiple user IDs share a host, and in this case u_1 and u_2 both map to one common host (e.g., used by two family members); (2) IP$_i$ is a proxy associated with multiple hosts; (3) u_2 is a guest user (either a real guest account, or introduced by an attacker) to the host associated with u_1, or vice versa. The challenge is how to differentiate the three cases based on available observations. During this process, we would like to maximize the number of tracked events with as few inconsistent events as possible.

More generally, Figure 3 shows the processing flow of HostTracker. At all times, HostTracker maintains a set of ID groups. At the very beginning, HostTracker uses a probabilistic model to initialize the set of tracked ID groups by grouping user IDs that were highly correlated in their login patterns (Section 4.2). HostTracker then applies an iterative refinement process to update the estimated model. HostTracker constructs a host-tracking graph using the current estimations and detects inconsistencies (Section 4.3). Later, HostTracker applies a separate process to remove inconsistencies (Section 4.4). Finally, HostTracker updates the ID groupings, and the updated ID groups are then fed back into the graph construction process again in an iterative way, until the estimated model converges (Section 4.5). The final output is a host-tracking graph with the corresponding identity-mapping table.

4.2 Application-ID Grouping

To derive host-IP bindings using application IDs, our first step is to compute an initial set of ID groups so that each group, with high probability, is associated with one common host. To do so, we compute the probability of multiple user IDs logging in nearby in time from a common host based on their login events.

Intuitively, if a group of users all logged in from one host, their login events will show strong correlations, i.e., appearing at a similar set of IP addresses at nearby times. For example, if user u_1 logged in 10 times at 10 different IP addresses, and u_2 logged in 9 times, and among these login events, user u_1 and u_2 appeared 8 times next to each other at 8 different IP addresses, then they are very likely to share one host.

However, by random IP address assignment, two unrelated users might also login consecutively if they happened to have connected to the Internet consecutively at a common IP address. In particular, in the proxy case, two different users repeatedly using a common proxy might also have a large number of close-by login events.

To quantitatively compute the probability of two independent user IDs u_1 and u_2 appearing consecutively, let us assume that each host's connection (hence the corresponding user login) to the Internet is a random, independent event. Given a sequence of login events from a user u_1, the number of times another user u_2 logged in right before or after u_1 in time follows a binomial distribution. Specifically, let n_1 denote the number of neighboring login slots to u_1, then the probability of u_2 appearing k or more times in these slots can be approximated as follows:

$$P(u_1, u_2) = 1 - \sum_{i=0}^{k-1} \binom{n_1}{i} p_2^i (1-p_2)^{n_1-i}$$

Here p_2 is the probability of u_2 logging in. With a total of c login events and a total of c_2 login events from u_2, we can approximate p_2 by $\frac{c_2}{c}$. A very small $P(u_1, u_2)$ means there is very little chance for u_1 and u_2 to log in consecutively k times if they are independent, suggesting that u_1 and u_2 might be correlated instead.

HostTracker identifies all pairs of users who have consecutive logins for every IP address. If a pair of user (u_1, u_2) have at least two consecutive logins across all IP addresses in a range (regardless whether the two consecutive logins happened at a same IP address or at two different IP addresses), HostTracker performs this correlation test for this pair, and selects (u_1, u_2) as a correlated user ID pair if $P(u_1, u_2)$ is smaller than a pre-set threshold. For example, if user u_2 logged in 62 times out of 133 of total logins, then $p_2 = 0.47$. In this case, if u_1 and u_2 logged in together 9 times out of a total of $n_1 = 49$ neighboring slots for u_1, then the probability of them appearing together in a random assignment is as high as 0.87. Hence we should not group the two users in this case. Currently, we set the probability threshold to 0.05, but the threshold could perhaps be more systematically tuned based on the login event patterns from each IP range.

Once HostTracker identifies all pairs of correlated users, it further groups them. If user ID pair (u_1, u_2) and (u_2, u_3) both are correlated pairs, then HostTracker groups all three users into a correlated user set $\{u_1, u_2, u_3\}$. This process continues, until there are no more groups that can be further expanded.

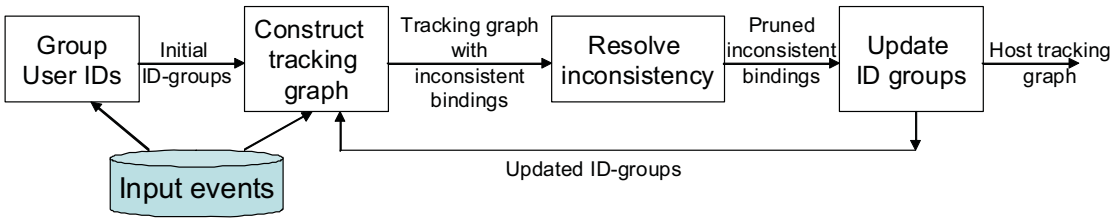

Figure 3: The processing flow of HostTracker.

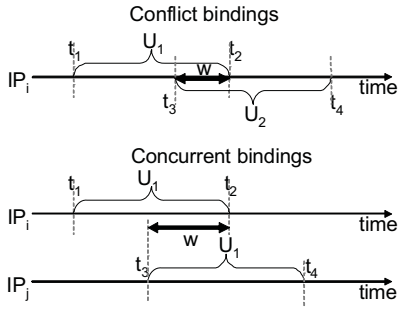

Figure 4: Examples of conflict bindings and concurrent bindings.

At this stage, each expanded ID group is regarded as tracked. For the remaining user IDs that cannot be grouped, HostTracker regards each user as a tracked ID group if the user has logged in at least twice throughout the trace duration. Both sets are merged together for further analysis. In the next few steps, HostTracker iteratively prunes and updates this set.

4.3 Host-Tracking Graph Construction

HostTracker regards each ID group as a candidate host identifier. It constructs a host-tracking graph using these groups. The first operation of this step is relatively simple: for each group U, HostTracker finds the first timestamp t_1 and the last timestamp t_2 at which any user from U logged in at IP address IP_i, and lets $w = [t_1, t_2]$ be the binding window for U at IP_i. We indicate the binding by $G(U, w) = IP_i$.

The second operation is to mark all inconsistent bindings on the graph. There are two types of inconsistent bindings (shown in Figure 4):

- **Conflict bindings:** Two user groups concurrently used the same IP address. For any two bindings $G(U_1, w_1) = IP_i$ and $G(U_2, w_2) = IP_i$, where $w_1 = [t_1, t_2]$ and $w_2 = [t_3, t_4]$. If $t_3 \leq t_2$ and $t_1 \leq t_4$, HostTracker first identifies the overlapped time range w between w_1 and w_2. It then marks both $G(U_1, w) = IP_i$ and $G(U_2, w) = IP_i$ as conflict bindings.
- **Concurrent bindings:** A user group concurrently used two different IP addresses. Specifically, consider two bindings $G(U_1, w_1) = \text{IP}_i$, where $w_1 = [t_1, t_2]$, and $G(U_1, w_2) = \text{IP}_j$, where $w_2 = [t_3, t_4]$. If $\widehat{\text{IP}}_i \neq IP_j$ and w_1 and w_2 overlap, then similarly to the conflict binding case, HostTracker finds the overlapped time range w, and marks both $G(U_1, w) = IP_i$ and $G(U_1, w) = IP_j$ as concurrent bindings.

In the next section, we describe how HostTracker iteratively resolves these inconsistent bindings.

4.4 Resolving Inconsistency

Inconsistent bindings can be caused by the existence of NATs and proxies, guest login events from IDs that cannot be eventually tracked, or incorrect groupings. In this section, we go through these three cases sequentially.

4.4.1 Proxy Identification

Proxy identification can resolve conflict bindings. Since large proxies or NATs allow hosts to concurrently access the Internet through them, they will generate a large number of conflict bindings. Depending on the network configurations, proxies/NATs can use either static or dynamic IP addresses. For example, a DSL-based Internet café may change its IP addresses from time to time.

To find both types of proxies/NATs, HostTracker gradually expands all the overlapped conflict binding windows associated with a common IP address. The purpose of such expansion is to obtain a maximum continuous *conflict window* with a large number of conflict bindings. Specifically, if $G(U_1, w_1) = \text{IP}_i$ and $G(U_2, w_2) = \text{IP}_i$, and the conflict windows w_1 and w_2 overlap, then HostTracker merges the two windows into an expanded conflict window.

For each expanded conflict window, HostTracker checks the degree of event concurrency. It attributes the conflict window to a proxy if the rate of new user arrivals is greater than α and the rate of conflict bindings is higher than β. In our current implementation, we empirically set α to 1 new user per 5 minutes and β to 1 conflict binding per 30 minutes.

After HostTracker identifies a binding window for a proxy, it marks all events falling into the window as proxy events. Recall that we treat proxy events as tracked events, with the corresponding proxy as their origin. With the limitations of our datasets, we do not further distinguish individual hosts using the proxies. In practice, this step can reduce the number of conflict bindings by more than 90%.

4.4.2 Guest Removal

Guest removal is helpful for resolving conflict bindings too. After we mark the user IDs as potentially "tracked" or "untracked" in our initial estimation, it is possible that a tracked group and an untracked ID concurrently appear at the same IP address, resulting in conflict bindings. There are three possibilities in this case: (1) the login of an untracked user ID is a guest login event, and in this case the tracked user group represents the correct host; (2) the untracked ID and the tracked group share a host, but they were not grouped in our first step because of the infrequent logins from the untracked ID; (3) in contrast to case (1), the login event of the tracked user group is a guest event.

In this case, HostTracker uniformly treats events from the untracked group as guest events to resolve the conflict. In both (1) and (2), the guest login events from the untracked ID will be cor-

rectly attributed to the host corresponding to the tracked group. In practice, we found (3) to be rare.

Overall, conflict bindings caused by guest events represent only a small fraction of the total inconsistent events. Although guest events are relatively few (see Section 5.2), it is still important to identify the hosts responsible for them. In our applications, we found most of the guest events to be malicious events on compromised hosts.

4.4.3 *Splitting Groups*

Splitting groups can resolve concurrent bindings. If a user ID group has a large number of concurrent bindings, the initial grouping step might be overly aggressive in grouping IDs from different hosts together. So for each group that had more than one concurrent binding, HostTracker adjusts the grouping by splitting the subset of IDs that caused concurrent login events into a different ID group. For each concurrent binding, HostTracker splits the group and examines whether there still exists concurrent bindings afterwards. If so, it continues the splitting process until there exists no more concurrent bindings for the IDs in the original group. In practice, concurrent bindings are a small portion of all the conflicts, and not many groups are affected in this step.

4.5 Closing the Loop

With the knowledge of proxies, guest events, and the split groups, HostTracker re-estimates the initial identity mappings by removing groups whose member IDs correspond to proxy-only users and guest-only users. Therefore, these IDs become untracked.

This process may appear straightforward. However, the order of the pruned users will affect the final set of remaining tracked IDs. If HostTracker incorrectly identified a trackable user as a guest, then all its events would be regarded as untracked. Given that our goal is to maximize the number of tracked events, only groups appearing at proxies are pruned initially. HostTracker can then iteratively refine the remaining groups following the steps of Section 4.4. In practice, for most IP ranges, the number of tracked groups converges after 4 to 6 iterations.

At this point, it is difficult to further resolve the remaining inconsistent bindings without additional information. To be conservative, HostTracker discards all the events from these inconsistent bindings and treats them as untracked events.

For all the identified host-IP bindings, their actual binding windows may be much longer than the binding windows derived from the login events only. So HostTracker expands their window sizes to increase the coverage. As illustrated by Figure 5, for a host-IP binding $G(A, [t_1, t_2]) = \text{IP}_i$, its window expansion is subject to two constraints: it cannot expand beyond the boundaries of the previous and next binding windows of the same host, nor can it expand beyond the boundaries of the previous and next binding windows on the same IP address. Under the two constraints, HostTracker opportunistically increases the binding-window size by one hour both forward and backward in time. In the case where the time between neighboring events is smaller than 1 hour, HostTracker divides the time gap equally to expand both binding windows.

Finally, HostTracker outputs both the identity-mapping table and the host-tracking graph with expanded binding windows. It classifies Events associated with the identified host-IP bindings are classified as tracked events. For host-IP bindings of a proxy, the corresponding events can be attributed to the proxy that originated the traffic. The remaining untracked events are either non-guest events from untracked IDs, or events within the unresolved inconsistent bindings (those discarded).

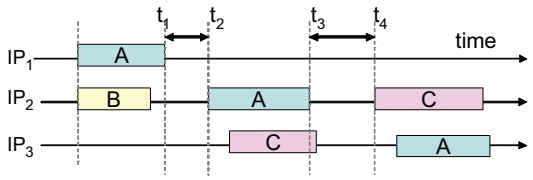

Figure 5: An example of binding-window expansion, where the host-IP binding is $G(A, [t_2, t_3]) = IP_2$. The window can maximally expand from t_1 to t_4.

5. HOST-TRACKING RESULTS AND VALIDATIONS

We applied HostTracker on traces collected at a major Web-email service provider. This section presents the host-tracking results using this dataset. We also evaluate the results using a separate software-update dataset that contains information on the actual host hardware IDs. Our study focuses on understanding the following aspects of tracking host-IP bindings:

- **Coverage:** What percentage of events or time duration can be associated with tracked hosts? How many hosts can be tracked at each network? We find that a large percentage of events can be tracked. The high percentage of tracked events is consistent across most of the IP ranges.

- **Accuracy:** How accurate are the bindings between IP addresses and hosts? Even with dynamic IP addresses and proxies being widely deployed, HostTracker achieves 92%–96% accuracy in our study.

- **Trackable user characteristics:** Can users be tracked across different locations? We find that the majority of the users can be tracked in only one or two network ranges. Highly mobile users that cannot be tracked at all are more suspicious in their email-sending patterns.

5.1 Input Dataset

The input dataset to HostTracker is a month-long user-login trace collected at a large Web-email service provider in October, 2008. The input data volume is about 330 GB, with each entry having three fields: (1) an anonymized user ID, (2) the IP address that was used to perform email login, and (3) the timestamp of the login event. In the log, we observe over 550 million unique user IDs and more than 220 million unique IP addresses. A large percentage of this user population consists of home users that access the Internet from dynamic IP-address ranges.

To obtain IP-address range information, we used the BGP-prefix table collected for the same period. For host-tracking to be statistically meaningful, we discard inactive IP ranges with fewer than 100 events or active for fewer than 7 days during the entire month, and we apply HostTracker to the remaining 30K ranges.

To evaluate the host-tracking results, ideally we would like to obtain the ground truth of the actual host-IP bindings. Without such information available, we adopt a month-long software-update log collected by a global software provider during the same period of October, 2008. The data entries we used for our validation include a unique hardware ID for each remote host that performs an update, the IP address of the remote host, and the software update timestamp.